Program translation for C program for software development

Patent Number: DE19617719
Publication date: 1997-11-13
Inventor(s): KRIEGEL URICH DR RER NAT (DE)
Applicant(s): FRAUNHOFER GES FORSCHUNG (DE)
Requested Patent: DE19617719
Application Number: DE19961017719 19960430
Priority Number(s): DE19961017719 19960430
IPC Classification: G06F9/44
EC Classification: G06F9/44G2
Equivalents:

-

Abstract

A C++ language program consists of source data files, whose source data files are to be translated in separate passages. Templates are assigned to code by instantiation, into an object code which is converted by ensuring links into a workable code. The translation of the source data files into object code data files and code generation of instantiation of the templates is carried out into separate translation passages, in which templates are instantiated by using all the source files of the programs.

---

(19) BUNDESREPUBLIK DEUTSCHLAND

DEUTSCHES PATENTAMT

(12) **Offenlegungsschrift**

(10) **DE 196 17 719 A1**

(51) Int. Cl.[6]: G 06 F 9/44

(21) Aktenzeichen: 196 17 719.7
(22) Anmeldetag: 30. 4. 96
(43) Offenlegungstag: 13. 11. 97

(71) Anmelder:
Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V., 80636 München, DE

(74) Vertreter:
Anwaltskanzlei Münich, Rösler, Steinmann, 80689 München

(72) Erfinder:
Kriegel, Urich, Dr.rer.nat., 12621 Berlin, DE

(56) Entgegenhaltungen:
US 53 75 242
EP 03 71 944 A2

Prüfungsantrag gem. § 44 PatG ist gestellt

(54) Verfahren zur Programmübersetzung eines in der Programmiersprache C++ geschriebenen Programms

(57) Beschrieben wird ein Verfahren zur Programmübersetzung eines in der Programmiersprache C++ geschriebenen, aus mehreren Quelldateien bestehenden Programms, dessen Quelldateien insbesondere zur Software-Entwicklung mehrmals, in getrennten Durchgängen zu übersetzen sind und Templates benutzen, die durch Instantiierung einem Code zugewiesen werden, in einen Objektcode, der durch anschließendes Linken in einen ausführbaren Code umgesetzt wird.
Die Erfindung zeichnet sich dadurch aus, daß die Übersetzung der Quelldateien in Objektcode-Dateien und die Codeerzeugung für die Instantiierung der Templates in getrennten Übersetzungsdurchgängen durchgeführt wird, wobei für die Instantiierung der Templates alle Quelldateien des Programms benutzt werden.

DE 196 17 719 A1

DE 196 17 719 A1

Beschreibung

Die Erfindung bezieht sich auf in Verfahren zur Programmübersetzung eines in der Programmiersprache C++ geschriebenen, aus mehreren Quelldateien bestehenden Programms, dessen Quelldateien insbesondere zur Software-Entwicklung mehrmals, in getrennten Durchgängen, zu übersetzen sind und Templates benutzen, die durch Instantiierung einem Cod zugewiesen werden, in einen Objektcode, der durch anschließendes Linken in einen ausführbaren Code umges tzt wird.

Bei der Programm- bzw. Software-Entwicklung wird in üblicher Weise ein Quellprogramm vom Programmierer erstellt, das in einer höheren Programmiersprach geschrieben ist. Derartige höhere Programmiersprachen, auch Hochsprachen genannt, verfügen über einen prozessorunabhängigen Befehlssatz, der zur Weiterverarbeitung in einem Computer zunächst in eine maschinenlesbare Sprache übersetzt werden muß. Für diesen Übersetzungsschritt bedient man sich spezieller Übersetzungsprogramme, sogenannte Compiler-Programme, die das Quellprogramm vollständig einmal und dauerhaft übersetzen, wobei zur vollständigen Übersetzung des Gesamtquellprogramms meist mehrere Übersetzungsdurchgänge notwendig sind. Durch diesen sogenannten Compilierungsvorgang wird ein Objektcode erhalten, der zur fehlerfreien Anwendung in einem Rechner noch mit entsprechenden sogenannten Bibliotheken verbunden werden muß. Dies erfolgt mit Hilfe eines Linkers, der ebenfalls ein Programm darstellt und den mit einem Compiler generierten Objektcode mit den Bibliotheken zu einem ablauffähigen Programm verbindet.

Unter der Vielzahl bekannter höherer Programmiersprachen beschränken sich die folgenden Ausführungen ausschließlich auf die Programmiersprache C++, die im Jahre 1980 als Weiterentwicklung der Programmiersprache C hervorging. Zwar gilt sie als assemblernahe Programmiersprache, d. h. es werden maschinennahe Anweisungen verwendet, doch sind die in C++ verwendeten Anweisungen prozessorunabhängig, so daß ein in C++ geschriebenes Programm wie vorstehend beschrieben in einen maschinenlesbaren Code übersetzt werden muß.

Neben den in der Programmiersprache C++ geschriebenen Anweisungen, aus denen die einzelnen Quelldateien eines in der C++-Sprache geschriebenen Programmes bestehen, sehen C++-Programme sogenannte Templates vor, die als Platzhalter für näher zu spezifizierende Informationen im Programm integriert sind. Templates ermöglichen es beispielsweise Implementationen von Klassen und Funktionen zu parametrisieren. Derartige, als Templates bezeichnete Platzhalter, werden häufig in C++-Programmen benutzt.

Eine besondere Eigenart von Templates besteht darin, daß bei der Compilation der in der C++ -Sprache geschriebenen Quelldateien, in denen Templates enthalten sind, für jedes Template ein Code erzeugt wird, der in einer vom Compiler zu erzeugenden, sogenannten Template-Include-Datei abgespeichert wird. Dieser Vorgang der Codeerzeugung für Templates wird auch als Instantiierung bezeichnet und ist für den nachfolgenden Linkvorgang, wie im weiteren beschrieben wird, von entscheidender Bedeutung. Zu Beginn des Link-Prozesses für C++-Programme wird zunächst geprüft, ob von bestehenden Template-Include-Dateien aktuelle Objektcode-Dateien existieren. Ist das nicht der Fall, so werden diese durch den C++ -Compiler erzeugt.

Je größer die Anzahl der im Quellprogramm vorhandenen Templates, umso mehr Codes werden bei der Instantiierung in die Template-Include-Datei eingeschrieben, wodurch die Template-Include-Datei an Speichervolumen zunimmt.

Hinzukommt, daß insbesondere im Stadium der Programm-Entwicklung Übersetzungsdurchgänge für einzelne Quelldateien mehrmals durchgeführt werden müssen, bei denen pro Übersetzungsdurchgang bei jedem auftretenden Template eine Codezuweisung erfolgt, die in der Template-Include-Datei abgespeichert wird. Auf diese Weise wächst das Speichervolumen der Template-Include-Datei sehr stark an.

Bei den bisher bekannten Compiler-Programmen für die Übersetzung von C++-Programmen werden die von Templates herrührenden Codes additiv in der generierten Template-Include-Datei abgespeichert, unabhängig davon, ob inhaltsgleiche Informationen bereits in der Datei enthalten sind oder nicht. Bei zyklischen Programmübersetzungsprozessen hat diese Vorgehensweise jedoch fatale Folgen: Zum einen werden die Template-Include-Dateien sehr groß und zum anderen wird das Compiler-Programm durch Hinzufügen identischer Informationen vor dem Linkvorgang der Objekt-Dateien gezwungen, die Informationen in der Template-Include-Datei neu zu übersetzen.

Aufgrund des großen Speichervolumens der sich im Rahmen mehrerer Entwicklungszyklen bildenden Template-Include-Datei bewegen sich die Zeitdauern für den nachfolgenden Linkprozeß in der Größenordnung von 20 bis 40 Minuten.

Der Erfindung liegt daher die Aufgabe zugrunde, ein Verfahren zur Programmübersetzung eines mit der Programmiersprache C++ geschriebenen, aus mehreren Quelldateien bestehenden Programms, dessen Quelldateien insbesondere zur Software-Entwicklung mehrmals, in getrennten Durchgängen, zu übersetzen sind und Templates benutzen, die durch Instantiierung einem Code zugewiesen werden, in einen Objektcode, der durch anschließendes Linken in einen ausführbaren Code umgesetzt wird, dadurch weiterzuentwickeln, daß die Größe der bei der Instantiierung von Templates erforderlichen Template-Include-Datei erheblich minimiert wird, so daß die Zeitdauer für das anschließende Linken wesentlich verkürzt werden kann.

Die Lösung der der Erfindung zugrundeliegenden Aufgabe ist im Anspruch 1 und 2 angegeben. Vorteilhafte Ausführungsformen sind den Ansprüchen 2 ff. zu entnehmen.

Erfindungsgemäß wird ein Verfahren zur Programm-Übersetzung eines in der Programmiersprache C++ geschriebenen, aus mehreren Quelldateien bestehenden Programms, gemäß des Oberbegriffs des Anspruchs 1, derart ausgebildet, daß die Übersetzung der Quelldateien in Objektcode-Dateien und die Codeerzeugung für die Instantiierung der Templates in getrennten Übersetzungsdurchgängen durchgeführt wird, wobei für die Instantiierung der Templates alle Quelldateien des Programms benutzt werden.

Insbesondere zeichnet sich das erfindungsgemäße Verfahren dadurch aus, daß bei der getrennten Überset-

zung der Quelldateien in den Obj ktcode solange kein Code für die Instantiierung der Templates erzeugt wird, bis alle Quelldateien übersetzt sind und daß im Anschluß daran in einem getrennten Übersetzungsvorgang unter Benutzung aller Quelldateien des Programms ein optimierter Code für die Instantiierung der Templates erzeugt wird, der während des nachfolgenden Linkens benutzt wird.

Der Erfindung liegt die Idee zugrunde, die in einem C++ -Programm enthaltenen Templates in einem einzigen Übersetzungsdurchgang zu übersetzen, der getrennt von den Übersetzungsdurchgängen für die Quelldateien abläuft. Insbesondere bei der Software-Entwicklung ist diese Vorgehensweis von besonderem V rteil, da der Inhalt der Template-Include-Dateien auch bei mehrmaligem Übersetzen gleicher Programmteile nicht modifiziert wird, wodurch der nachfolgende Link-V rgang in wesentlich kürzerer Zeit, d. h. in wenigen Minuten, vorzugsweise 2 bis 4 Minuten, und unabhängig von der Zahl der zuvor erfolgten Übersetzungsdurchläufe ablaufen kann.

Die im vorstehenden beschriebene erfindungsgemäße Idee ist anhand eines konkreten Forschungsprojektes der Anmelderin getestet worden. In einem Projekt "FAPU" ist in der Programmiersprache C++ ein Programmsystem realisiert worden, das in etwa 160000 Zeilen Code über etwa 100 mittels Templates realisierte parametrisierte Klassen enthält. Bei ständig wachsender Anzahl der Programmdurchläufe durch den andauernden Entwicklungszyklus traten auf einer IBM RS/6000, Typ 390 H mit 128 MB-Hauptspeicher reine Linkzeiten von etwa 40 Minuten auf. Zur Erklärung hierfür sei im folgenden eine Passage aus dem Programmübersetzungs-Benutzerhandbuch (xlC-Übersetzer, IBM 93) zitiert:

"By default, the compiler builds and compiles the special template-include files in the tempinc subdirectory of the working directory.

The compiler builds a template-include file corresponding to each header file containing template function declarations. After the compiler creates one of these files, it may add any information to it as each compilation unit is compiled. However, the compiler never removes information from the file".

Bei einem wie im Vorstehenden zitiert, Datei-basierten Arbeiten, ist dies die einzige mögliche Vorgehensweise. Der Compiler besitzt keinerlei sogenannte Kontextinformationen über die zu einer Anwendung gehörenden Dateien, so daß er nur Informationen zu den Beschreibungsdateien hinzufügen kann. Bei zyklischen Entwicklungsprozeß hat dies jedoch fatale Folgen.

Zum einen werden die Dateien im tempinc-Unterverzeichnis schnell sehr groß, und zum anderen wird der Compiler durch das Hinzufügen identischer Informationen vor dem eigentlichen Binden der Objektdateien gezwungen, die Dateien im tempinc-Unterverzeichnis neu zu übersetzen.

Als Endeffekt erhält man je nach der Anzahl der Entwicklungszyklen seit dem letzten kompletten Löschen des tempinc-Verzeichnisses reine Linkzeiten in der Größenordnung von 20 bis 40 Minuten. Die im Benutzerhandbuch vorgeschlagene Lösung, das tempinc-Unterverzeichnis zu löschen, kann jedoch nicht immer verwendet werden. Gerade bei großen Anwendungen kann dies nicht realisiert werden, da dann alle zur Anwendung gehörenden Dateien auch neu kompiliert werden müßten. Dies bedeutet, daß die Vorteile des Make-Mechanismus außer Kraft gesetzt würden und eine Verkürzung der Linkzeit durch eine längere Kompilationszeit erkauft wird.

Die prototypische Lösung im vorstehenden Projekt bestand darin, aus allen zu einem Programmsystem gehörenden Dateien eine spezielle "template-include-Datei" vor dem Linkprozeß zu generieren.

So kann dem verwendeten xlC-Compiler über einen "Schalter" mitgeteilt werden, in welchem Verzeichnis die Template-Include-Dateien angelegt werden sollen. Es ist jedoch nicht möglich, die Generierung dieser Dateien zu unterbinden, ohne daß die Codes für die Instantiierung der Templates "inline" kompiliert werden. Deshalb wird mit unterschiedlichen Verzeichnissen für die Template-Include-Dateien während der Übersetzungs- und der Linkphase gearbeitet. Die zur Kompilationszeit erfolgten Einträge werden anschließend gelöscht. Die eigentliche Datei zur Beschreibung der Templates — fapu.hh.C — wird vor der Link-Phase unabhängig vom verwendeten Compiler zunächst als temporäre Datei gemäß der in dem vorstehend genannten Benutzerhandbuch (IBM93) beschriebenen Struktur erzeugt. Dazu werden alle zu einem Projekt gehörenden Header- und Implementationsdateien parsiert. Um unnötige Kompilationen der Template-Beschreibungsfunktionen während der Link-Phase zu vermeiden, wird die temporäre im Template-Include-Datei nur dann in das Verzeichnis fapu-tempinc kopiert, wenn dort noch keine Datei fapu.hh.C im entsprechenden Verzeichnis existiert, oder wenn diese Unterschiede zu der temporären Version aufweist.

Das im folgenden angegebene Programm ist ein sogenanntes Shell-Programm und dient sowohl zur Generierung eines Datenbank-Schemas zur Benutzung mit der objektorientierten Datenbank "ObjektStore" (ODI94) als auch zur Generierung der Template-Include-Datei für den xlC-Compiler, da beide mit analogen Strukturen arbeiten. In dem vorstehend beschriebenen Projekt "FAPU" sind nur Templates für spezielle Arten von Objekt-Store-Collection-Klassen benutzt worden, wodurch die Programmstruktur sehr einfach gehalten werden konnte.

Der folgende Programmteil dient lediglich zur Angabe eines erfindungsgemäßen Beispieles zur Anwendung der technischen Lehre des vorstehend geschilderten erfindungsgemäßen Gedankens:

```
#!/bin/sh
#
#  create the schema source file for FAPU
#  call generate-schema.sh pdir pdf
#  pdir directory relative to workspace
#  pdf project description file

preserve=false
curws=$WORKSPACE
noschemaflag=true
interactive=false
message="Usage: $0 [-p] [-w workspace] [-o path] pdir pdf"
while getopts io:pw: c
do
case $c in
  i) interactive=true;;
  o) noschemaflag=false;schema=$OPTARG;;
  p) preserve=true;;
  w) curws=$OPTARG;;
  \?) echo $message
  exit 2;;
esac
done

shift `expr $OPTIND - 1`

AWK=/opt/GNU/bin/gawk
if  [ $# != 2 ]; then
    echo $message
    exit 1
fi
pdir=$1
pdf=$2

dependency=$curws/$pdir/.sniffdir/schema.incl
if $noschemaflag; then
  schema=$curws/$pdir/os.schema.C
fi
```

```
if [ -z „$curws" ]; then
    echo „$curws has not been set" >&2
    exit 1
fi

if [ -z „$SHARED_SRC" ]; then
    echo „$SHARED_SRC has not been set" >&2
    exit 1
fi

#  redirect output into schema source file
#

if $preserve; then
  if [ -f $schema ]; then
   oldschema=true
   mv -f  $schema $schema~
  else
   oldschema=false
  fi
else
  oldschema=false
  rm -f $schema
fi

# -get-alle-source-files in project
#
sources=
#estimate true path of project files
#look in $curws first. If the file is located there then ok, else assume
#it is located in $SHARED_SRC . If it is not there. print a warning and
#forget it
for file in `get-project-files.pl -w $curws $pdir $pdf | grep
„\.[h|hh|H|C]$"`
  do
  if [ -r „$curws/$file" ]; then
    sources="$sources" „$curws/$file
  else
   if [ -r „$SHARED_SRC/$file" ]; then
     sources="$sources" „$SHARED_SRC/$file
   else
     echo „%%%Warning: File ${file} is not readable and will therefore not
contribute to schema generation"
   fi
  fi
done


#  analyze source files and find persistent classes and their
#  headers
#

eval `
    $AWK ,
   /class/ {
       for (i = 1; i <= NF; i++)
      if ($i == „class") { i++; class = $i; }
   }
```

```
    /static.*get_os_typespec/ {
        if (class != "") {
       headers = headers FILENAME " ";
       classes = classes class " ";
       persistent_classes[class]=class;
        }
        class = ""
       }
    /::get_os_typespec()/ {
        for (i = 1; i<=NF; i++) {
             if ((ma=match($i,"::get_os_typespec")) > 0) {
               instance=substr($i,1,ma -1 );
               instances[instance]=instance; } }
     }
    END {
        for (ins in instances)
          {
          if (persistent_classes[ins] != ins) {
          wrong_instances= wrong_instances ins " ";}
         }
        printf "headers=\"%s\"\n", headers;
       printf "classes=\"%s\"\n", classes;
       printf "wrong_instances=\"%s\"\n", wrong_instances;

    }
     , $sources
`
if [ ! -z "$wrong_instances" ]; then
  if $interactive; then
   $PAPU_ENV/message.tcl "The following classes have no 'static
get_os_typespec()' but persistent instances are created" $wrong_instances
"The Usage of such a schema is on your own RISK!"
  fi
  echo "The following classes have no  'static get_os_typespec()' member
function but it was tried to create persistent instances:
${wrong_instances}"
  echo "The Usage of such a schema is on your own RISK!"

fi



# start writing output in schema file

exec > $schema
#  output header-inclusions
#
cat <<EOF
//
// ObjectStore Schema Source File
// PAPU-Projekt
//
//      Generated: `date`
#ifdef OS_SCHEMA
#include <ostore/manschem.hh>
#else
//
// absolute Pfade für Template-Defs to fake xlC
```

```
//
/*9999999999*/#include „/usr/lpp/xlC/include/ostore/coll/coll_pt.hh"
/*0000000000*/#include „/usr/lpp/xlC/include/ostore/coll/coll_pt.c"
/*9999999999*/#include „/usr/lpp/xlC/include/ostore/coll/list_pt.hh"
/*0000000000*/#include „/usr/lpp/xlC/include/ostore/coll/list_pt.c"
/*9999999999*/#include „/usr/lpp/xlC/include/ostore/coll/set_pt.hh"
/*0000000000*/#include „/usr/lpp/xlC/include/ostore/coll/set_pt.c"
#endif /*OS_SCHEMA*/
//
// mit Rose erzeugte Klassendefinitionen
//

EOF

echo $headers | tr , , ,\012' | sort | uniq | sed -e ,
        /^$/  d
        s%'$SHARED_SRC'/src/%%
        s%'$curws'/src/%%
        s/^\(.*\)$/\/\*9999999999\*\/ #include „\1"/
,

#  find all subclasses of ObjectStore template classes
#  and mark them to the compiler
#
cat <<EOF

// persistente Klassen, die vom ObjectStore templates abgeleitet
// wurden, fuer den xlC deklarieren
//

#ifdef __IBMCPP__
EOF

sed -e ,
   /os_List<.*>/b doit
   /os_Set<.*>/b doit
   /os_Collection<.*>/b doit
            d
            : doit
            s/ //g
            s/.*\(os_[A-Za-z]*<[^>]*>\).*/\#pragma define(\1)/
, $sources /dev/null | sort | uniq

echo „#endif"

#  some classes must be manually handled
#
cat <<EOF

//
// private Template-Klassen müssen manuell markiert werden
//

#include <os.schema.h>

EOF

#  mark all classes in project as persistent
```

```
#

cat <<EOF

#ifdef OS_SCHEMA

//
// persistente Klassen fuer ObjectStore markieren
//

void dummy(void)
{
EOF

echo $classes | tr , , ,\012' | sort | uniq | sed -e.,
        /^$/  d
        s/\(.*\)/    OS_MARK_SCHEMA_TYPE(\1);/
,

echo „}"

cat <<EOF
#endif /*OS_SCHEMA*/
EOF

message="New schema file ${schema}"
if $preserve; then
  if $oldschema; then
   differ="`diff -e  $schema $schema-|sed -e ,
          /^\/\/[\t ]*Generated/d
         /^[0-9][0-9]*c$/d
         /\./d
         q'`"
    if [ -z „$differ" ]; then
     mv $schema- $schema
     echo „Old schema file is already up to date." >&2
   else
     echo $message >&2
   fi
  else
   echo $message >&2
  fi
else
  echo $message >&2
fi

# Generate schema dependency file
if [ -r „$dependency" ]; then
  rm -f $dependency
fi

echo „#----Schema dependencies generated `date`" > $dependency
echo „SCHEMA_DEPENDENCY= \\" >> $dependency
echo $headers |tr , , ,\012'| sort | uniq | sed -e ,
        /^$/  d
        s%'$SHARED_SRC'/src/%%g
        s%'$curws'/src/%%g
        s/^\(.*\)$/\1/g
        , |tr ,\012' , , >> $dependency



  echo „" >> $dependency
  echo „#end" >> $dependency
```

Patentansprüche

1. Verfahren zur Programm-Übersetzung eines in der Programmiersprache C++ geschriebenen, aus mehreren Quelldateien bestehenden Programms, dessen Quelldateien insbesondere zur Software-Entwicklung mehrmals, in getrennten Durchgängen zu übersetzen sind und Templates benutzen, die durch Instantiierung einem Code zugewiesen werden, in einen Objektcode, der durch anschließendes Linken in einen

ausführbaren Code umgesetzt wird, **dadurch gekennzeichnet**, daß die Übersetzung der Quelldateien in Objektcode-Dateien und di Codeerzeugung für die Instantiierung der Templates in getrennten Übersetzungsdurchgängen durchgeführt wird, w bei für die Instantiierung der Templates alle Quelldateien des Programms benutzt werden.

2. Verfahren zur Programm-Übersetzung eines in der Programmiersprache C++ geschriebenen, aus mehreren Quelldateien bestehenden Programms, dessen Quelldateien insbesondere zur Software-Entwicklung mehrmals, in getrennten Durchgängen zu übersetzen sind und Templates benutzen, die durch Instantiierung einem Code zugewiesen werden, in einen Objektcode, der durch anschließendes Linken in einen ausführbaren Code umgesetzt wird, dadurch gekennzeichnet, daß bei der getrennten Übersetzung der Quelldateien in den Objektcode solange kein Code für die Instantiierung der Templates erzeugt wird, bis alle Quelldateien übersetzt sind und daß im Anschluß daran in einem getrennten Übersetzungsdurchgang unter Benutzung aller Quelldateien des Programms ein optimierter Code für die Instantiierung der Templates erzeugt wird, der während des nachfolgenden Linkens benutzt wird.

3. Verfahren nach Anspruch 1 oder 2, dadurch gekennzeichnet, daß der für die Programmübersetzung verwendete Übersetzer ein xlC-Übersetzer ist.

4. Verfahren nach einem der Ansprüche 1 bis 3, dadurch gekennzeichnet, daß eine Template-include-Datei generiert wird, in die die im Quellprogramm enthaltenen Templates in übersetzter Code-Form eingeschrieben werden.

5. Verfahren nach Anspruch 4, dadurch gekennzeichnet, daß bei einer erneuten Programmübersetzung eines bereits übersetzten Quellprogramms die erzeugte temporäre Template-include-Datei mit der bereits bestehenden Template-inlcude-Datei verglichen wird und bei Übereinstimmung übernommen wird.

6. Verfahren nach Anspruch 5, dadurch gekennzeichnet, daß bei der Übernahme der Erzeugungszeitpunkt der alten Template-include-Datei auf einen aktuellen Übersetzungszeitpunkt gesetzt wird, so daß diese Datei nicht noch einmal übersetzt wird.

– Leerseite –